МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра «Вычислительные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине

Аппаратное и программное обеспечение роботизированных систем

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гай В.Е.

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сипаков В.В.

(подпись) (фамилия, и.,о.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(шифр группы)

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2022

# Цель работы

Получение навыков работы с алгоритмами управления роботами

# Задание

Задание: выполнить вариант и загрузить программу на платформу для соревнований, записать видео работы, подготовить отчёт с подробным описанием результатов. В отчёт вставить результаты с соревнования. Есть примеры, можно на них посмотреть (видео на сайте).

1. **Обход препятствий**

<https://robotbenchmark.net/benchmark/obstacle_avoidance>

1. **Движение по квадрату**

<https://robotbenchmark.net/benchmark/square_path/>

# Алгоритм

## Обход препятствий

Использован следующий алгоритм: чтобы заметить препятствие, используются показания сенсоров. Чем ближе находится поверхность, на который смотрит сенсор, тем больше будут показания сенсора.

Таким образом, если сенсоры, которые установлены на левой части робота увеличивают значение, то нужно повернуть направо, для этого мы замедляем правое колесо, тем самым позволяя роботу совершить этот маневр. Соответственно, так же мы поступаем и если видим препятствие справа: замедляем левое колесо, чтобы совершить поворот.

Робот не знает, где финиш, поэтому выполняя множество поворотов, он может потеряться в пространстве. Для того, чтобы после каждого обхода препятствий робот выравнивался и ехал непосредственно к финишу, используется компас. Скорость выравнивания робота по компасу сделана специально небольшой, что увеличивает время достижения финиша, но дает нам гарантию того, что при повороте робот не врежется в препятствие, которое только что объезжал.

## Движение по квадрату

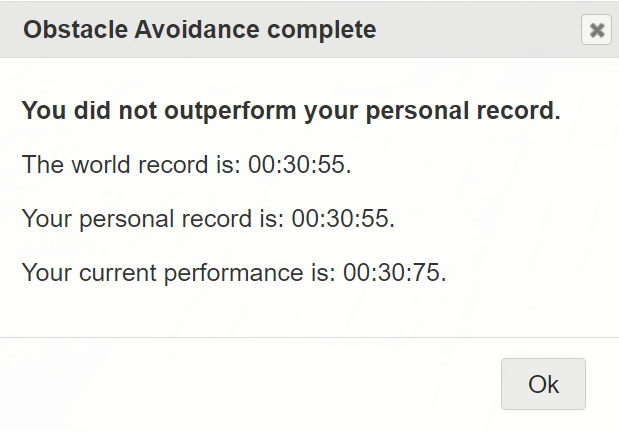
Алгоритм этой программы был представлен следующим образом: метод прохода первого сегмента был самый простой, так как все, что нам нужно было сделать – это проехать по прямой. Длину прямой мы узнали при помощи нескольких пробных экспериментов и использования сенсора на правом колесе, который нам и показал длину прямой. Это значения мы приняли за константу, чтобы работать с ним на оставшихся сегментах, так как у нас квадрат, а, как известно, его стороны равны. После каждого поворота происходит корректировка этой «константы», так как на повороте ее значение изменялось в меньшую сторону, ведь правое колесо немного отъезжало назад. Не стопроцентная точность была получена только лишь из-за поворотов, а именно из-за невозможности повернуть робота ровно на 90 градусов. Углы, на которые робот поворачивал, просто были подобраны так, что бы отклонение от ожидаемого маршрута было минимальным.

# Результат работы

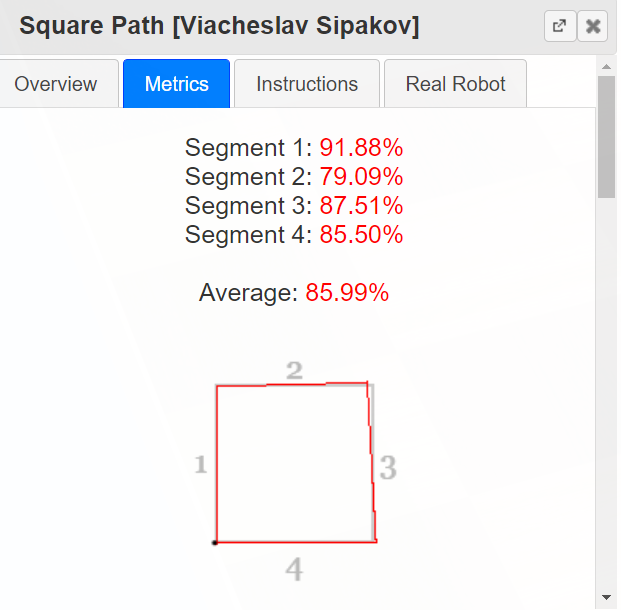
1. Результат (30:55), отображающийся в таблице лидеров, получен случайным образом на черновой версии контроллера, не имеющей компаса.



После этого получен еще один результат (~30-34 секунды), достаточно стабильный и близкий к лучшему. В нем уже присутствует обработка показаний компаса.



1. Результат: 86%





**avoid\_obstacles.py:**

sensors\_dict = {  
 0: 'outer left',  
 1: 'front left',  
 2: 'front',  
 3: 'front right',  
 4: 'outer right',  
}  
  
# You may need to import some classes of the controller module. Ex:  
# from controller import Robot, Motor, DistanceSensor  
from controller import Robot, Compass  
  
# create the Robot instance.  
robot = Robot()  
  
# get the time step of the current world.  
timestep = int(robot.getBasicTimeStep())  
  
# Constants of the Thymio II motors and distance sensors.  
maxMotorVelocity = 9.53  
distanceSensorCalibrationConstant = 360  
  
# get robot's Compass device  
compass = robot.getCompass('compass')  
  
leftMotor = robot.getMotor('motor.left')  
rightMotor = robot.getMotor('motor.right')  
  
sensors = [robot.getDistanceSensor('prox.horizontal.0'), robot.getDistanceSensor('prox.horizontal.1'),  
 robot.getDistanceSensor('prox.horizontal.2'), robot.getDistanceSensor('prox.horizontal.3'),  
 robot.getDistanceSensor('prox.horizontal.4')]  
  
# Set ideal motor velocity.  
initialVelocity = maxMotorVelocity  
  
for sensor in sensors:  
 sensor.enable(timestep)  
  
compass.enable(timestep)  
  
leftMotor.setPosition(float('inf'))  
rightMotor.setPosition(float('inf'))  
  
initialVelocity = maxMotorVelocity  
  
obstacleFlag = False  
obstacleCounter = 0  
  
while robot.step(timestep) != -1:  
 # Read the sensors:  
  
 compassData = compass.getValues()  
  
 sensorData = []  
 for sensor in sensors:  
 sensorData.append(sensor.getValue() / distanceSensorCalibrationConstant)  
  
 # Process sensor data here.  
  
 i = 0  
  
 print(sensorData)  
  
 if obstacleFlag:  
  
 stillObstacleFlag = False  
 for value in sensorData[1:-1]:  
 if value != 0:  
 print('still obstacle')  
 stillObstacleFlag = True  
 obstacleCounter = 4  
 break  
  
 if sensorData[0] > 8:  
 print('still obstacle left')  
 stillObstacleFlag = True  
 obstacleCounter = 4  
 break  
  
 if sensorData[4] > 8:  
 print('still obstacle right')  
 stillObstacleFlag = True  
 obstacleCounter = 4  
 break  
  
 if not stillObstacleFlag:  
 if obstacleCounter == 0:  
 obstacleFlag = False  
 print('no obstacle')  
 obstacleCounter -= 1  
 print(obstacleCounter)  
  
 # rounded by obstacles  
 if sensorData[0] != 0 and sensorData[1] != 0 and sensorData[2] != 0 and sensorData[3] != 0 and sensorData[  
 4] != 0:  
 rightMotor.setVelocity(-initialVelocity)  
 leftMotor.setVelocity(-initialVelocity)  
 print('going back (obstacle)')  
  
 # obstacle left  
 elif sensorData[0] != 0 or sensorData[1] != 0 or sensorData[2] != 0:  
 rightMotor.setVelocity(-initialVelocity)  
 leftMotor.setVelocity(initialVelocity)  
 print('turning right (obstacle)')  
  
 # obstacle right  
 elif sensorData[3] != 0 or sensorData[4] != 0:  
 rightMotor.setVelocity(initialVelocity)  
 leftMotor.setVelocity(-initialVelocity)  
 print('turning left (obstacle)')  
  
 # no obstacle  
 else:  
 if sensorData[0] < 1 and sensorData[4] < 1 and sensorData[1] == 0 and sensorData[2] == 0 and sensorData[3] == 0:  
 if round(compassData[0], 3) < (-0.01) and sensorData[0] == 0:  
 rightMotor.setVelocity(initialVelocity)  
 leftMotor.setVelocity(initialVelocity / 1.2)  
 print('turning left (compass)')  
 elif round(compassData[0], 3) > 0.01 and sensorData[4] == 0:  
 rightMotor.setVelocity(initialVelocity / 1.2)  
 leftMotor.setVelocity(initialVelocity)  
 print('turning right (compass)')  
 else:  
 rightMotor.setVelocity(initialVelocity)  
 leftMotor.setVelocity(initialVelocity)  
 else:  
 rightMotor.setVelocity(initialVelocity)  
 leftMotor.setVelocity(initialVelocity)  
  
 for value in sensorData[1:-1]:  
 if value != 0:  
 obstacleFlag = True  
 obstacleCounter = 4  
 print('obstacle:', sensors\_dict[i])  
 i += 1  
  
 if sensorData[0] > 8:  
 obstacleFlag = True  
 obstacleCounter = 4  
 print('obstacle:', sensors\_dict[i])  
  
 if sensorData[4] > 8:  
 obstacleFlag = True  
 obstacleCounter = 4  
 print('obstacle:', sensors\_dict[i])  
  
 pass

**square\_path.py:**

*"""Sample Webots controller for the square path benchmark."""*from controller import Robot  
  
# Get pointer to the robot.  
robot = Robot()  
  
# Get pointer to each wheel of our robot.  
leftWheel = robot.getMotor('left wheel')  
rightWheel = robot.getMotor('right wheel')  
  
# Get right wheel sensor  
rightWheelSensor = robot.getPositionSensor('right wheel sensor')  
rightWheelSensor.enable(16) # Refreshes the sensor every 16ms.  
  
# Repeat the following 4 times (once for each side).  
for i in range(0, 4):  
  
 # First set both wheels to go forward, so the robot goes straight.  
 leftWheel.setPosition(1000)  
 rightWheel.setPosition(1000)  
  
 if i == 0:  
 robot.step(3900)  
 else:  
 while rightWheelSensor.getValue() / (i + 1) < 19:  
 robot.step(1)  
 print(rightWheelSensor.getValue())  
  
 # Wait for the robot to reach a corner.  
 #robot.step(3900)  
  
 # Then, set the right wheel backward, so the robot will turn right.  
 leftWheel.setPosition(1000)  
 rightWheel.setPosition(-1000)  
 # Wait until the robot has turned 90 degrees clockwise.  
 if i == 0:  
 robot.step(464)  
 elif i == 1:  
 robot.step(464)  
 elif i == 2:  
 robot.step(465)  
 elif i == 3:  
 robot.step(465)  
  
# Stop the robot when path is completed, as the robot performance  
# is only computed when the robot has stopped.  
leftWheel.setVelocity(0)  
rightWheel.setVelocity(0)